Synthesis of Communicating Controllers for Distributed Systems

Gabriel Kalyon, Tristan Le Gall, Hervé Marchand, Thierry Massart

2011 50th IEEE Conference on Decision and Control and
European Control Conference (CDC-ECC)
Orlando, FL, USA, December 12-15, 2011

Synthesis of Communicating Controllers for Distributed Systems

G. Kalyon, T. Le Gall, H. Marchand and T. Massart

Abstract— **We consider the control of distributed systems composed of subsystems communicating asynchronously; the aim is to build local controllers that restrict the behavior of a distributed system in order to satisfy a global state avoidance property. We model our distributed systems as *communicating finite state machines* with reliable unbounded FIFO queues between subsystems. Local controllers can only observe their proper local subsystems and do not observe the queues. To refine their control policy, they can use the FIFO queues to communicate by piggybacking extra information to the messages sent by the subsystems. We define synthesis algorithms allowing to compute the local controllers. We explain how we can ensure the termination of this control algorithm by using abstract interpretation techniques, to overapproximate queue contents by *regular languages*. An implementation of our algorithms provides an empirical evaluation of our method.**

I. Introduction

In the framework of control of distributed systems, two classes of systems are generally considered, depending on whether the communications between subsystems are *synchronous* or not. When the synchrony hypothesis [3] can be made, the *decentralized control problem* and the *modular control problem* address the design of coordinated controllers that jointly ensure the desired properties for this kind of systems [26], [22], [21], [9], [13]. When considering *asynchronous* distributed systems, one have to take into account some communication delays between the components of the system, which renders the *distributed control problem* much harder even undecidable [24].

We are here interested in the second problem i.e., the distributed control problem. Our aim is to solve this problem when the system to be controlled is composed of n subsystems that asynchronously communicate through reliable *unbounded* FIFO channels (or queues). These subsystems are modeled by *communicating finite state machines* [5] (CFSM for short) that explicitly express the work and communications of a distributed system. This model appears to be essential for concurrent systems in which components cooperate via asynchronous message passing through unbounded buffers (they are e.g. widely used to model communication protocols). We thus assume that the distributed system is already built and the architecture of communication between the different subsystems is fixed. Following the architecture described in Figure 1, we assume that each subsystem is controlled by a *local* controller. A local controller observes the actions fired by its subsystem but cannot observe the contents of the queues. It communicates with its subsystem synchronously (we assume that both reside on the same site) and it is allowed to disable some of the actions of its subsystem; note that the set of actions of a subsystem is partitioned into the set of controllable (resp. uncontrollable) actions that can (resp. cannot) be forbidden by its controller. In our setting, we assume that each local controller keeps its own estimate of the current global state of the system and that some extra information can be piggybacked to the messages normally exchanged by the subsystems. These communications allow them to exchange information in order to refine their knowledge about the global state of the system by taking into account the estimates of the other controllers and thus their local control policy.

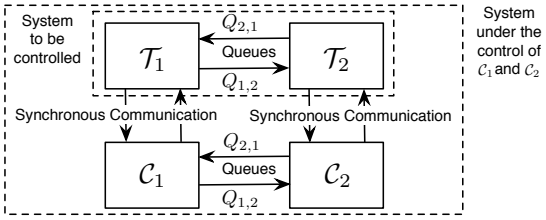

Fig. 1. Control architecture of a distributed system.

In this paper, we focus on the state avoidance control problem that consists in preventing the system from reaching some particular configurations. Note that, since the FIFO channels are unbounded, the state space of the distributed system to be controlled can be *infinite* and that the set of states that need to be avoided by control, can involve the contents of the channels that can not be observed by the controllers. To solve our control problem, it is necessary on one hand to compute offline, the set of states that leads to the forbidden states by only taking uncontrollable transitions and on the second hand to compute online, the states estimates for each controller. Our computation is based on the use of the (co-)reachability operator, which cannot always be done in the CFSM model for undecidability reasons. To overcome this obstacle, we rely on the abstract interpretation techniques we presented previously in [14]. They ensure the termination of the computations of our algorithm by overapproximating in a symbolic way the possible FIFO channel contents (and hence the state estimates) by regular languages.

Related Works. Over the past years a considerable research effort has been done in decentralized supervisory control [22], [26], [21], [11] that allows to synthesize individual controllers that have a partial observation of the system's moves and can communicate with each other [21],

This work has been done in the MoVES project (P6/39), part of the IAP-Phase VI Interuniversity Attraction Poles Programme funded by the Belgian State, Belgian Science Policy and was partly supported by the European Community's 7th Framework Programme under project DISC (DIstributed Supervisory Control of large plants), Grant Agreement INFSO-ICT-224498..

G. Kalyon and T. Massart are with U.L.B., Bruxelles, Belgium
T. Le Gall is with CEA LIST, LMeASI, 91141 Gif-sur-Yvette, France
H. Marchand is with INRIA, Rennes - Bretagne Atlantique, France

[1], [16]. In [8], Gastin *et al.* study the decidability of LTL synthesis depending on the architecture of the distributed system. However, in these works the authors consider a synchronous architecture between the controllers.

In [24], Tripakis studies the decidability of the existence of controllers such that a set of responsiveness properties is satisfied in a decentralized framework with communication delays between the controllers. He shows that the problem is undecidable when there is no communication or when the communication delays are unbounded. He conjectures that the problem is decidable when the communication delays are bounded. In [2], Bensalem *et al.* propose a knowledge-based algorithm for distributed control. Their approach is similar to our: each subsystem is controlled according to a (local) knowledge. Moreover, one can add synchronizations in order to get a better knowledge of the global system. Compared to [2], the reachability problem is undecidable in our framework and we do not add synchronizations. Moreover, state estimates are a kind of knowledge that does not depend on the property we have to ensure.

The control of concurrent systems is closely related to our framework [11], [9], [13], [15]. In this setting, the system is composed of several subsystems that communicate with each other synchronously. Each controller only observes the behavior of one subsystem and takes its control decision according to this knowledge. Compared with our approach, the different subsystems communicate synchronously as well as the controllers when such communications are allowed. In [7], Darondeau synthesizes distributed controllers for distributed system communicating by bounded channels. He states a sufficient condition allowing to decide if a controller can be implemented by a particular class of Petri nets that can be further translated into communicating automata.

Our problem differs from the synthesis problem (see e.g. [17], [10]) where the problem is to distribute the actions of a specification depending on the subsystem where they must be executed, and to synchronize them in such a way that the resulting distributed system is equivalent to the given global specification.

There are also some works dealing with the computation of a state estimate of a system with distributed controllers. For example, in [25], Xu and Kumar propose a distributed algorithm which computes an estimate of the current state of a system. Local estimators maintain and update local state estimates from their own observation of the system and information received from the other estimators. In their framework, the local estimators communicate between them through reliable FIFO channels with delays, whereas the system is monolithic, and therefore these FIFO channels are not included into the global states of the system.

Outline. The remainder of this paper is as follows. In section II, we define the formalism of *communicating finite state machines*, that we use to model distributed systems. We formally define, in section III, the control mechanisms and the state avoidance control problem. We define, in section IV, a control algorithm for our state avoidance control problem, and we explain in section V, how we can ensure the termination of this control algorithm by using abstract interpretation techniques. Section VI gives some experimental results.

II. Communicating Finite State Machines as Model of the System

We model a distributed system by the standard formalism of *communicating finite state machines* [5] which use reliable unbounded FIFO channels (also called *queues* below) to communicate. A *global state* in this model is given by the local state of each subsystem together with the content of each queue. Therefore, since no bound is given neither in the transmission delay, nor on the length of the queues, the global state space is a priori infinite.

Definition 1: [Communicating Finite State Machines] A communicating finite state machine *(*CFSM *for short)* $\mathcal{T}$ *is defined as a 6-tuple* $\langle L, \ell_0, Q, M, \Sigma, \Delta\rangle$*, where* (i) L *is a finite set of locations,* (ii) $\ell_0 \in L$ *is the initial location,* (iii) Q *is a set of queues that* $\mathcal{T}$ *can use,* (iv) M *is a finite set of messages,* (v) $\Sigma \subseteq Q \times \{!, ?\} \times M$ *is a finite set of actions, which are either an* output $i!m$ *to specify that the message* $m \in M$ *is written on the queue* $i \in Q$ *or an* input $i?m$ *to specify that the message* $m \in M$ *is read on the queue* $i \in Q$*, and* (vi) $\Delta \subseteq L \times \Sigma \times L$ *is a finite set of transitions.*

An output transition $\langle \ell, i!m, \ell'\rangle$ indicates that, when the system moves from the location ℓ to ℓ', a message m is added at the end of the queue i. An input transition $\langle \ell, i?m, \ell'\rangle$ indicates that, when the system moves from ℓ to ℓ', a message m must be present at the beginning of the queue i and is removed from this queue. Moreover, throughout this paper, we assume that $\mathcal{T}$ is deterministic, meaning that for all $\ell \in L$ and $\sigma \in \Sigma$, there exists at most one location $\ell' \in L$ such that $\langle \ell, \sigma, \ell'\rangle \in \Delta$. We assume that all queues are initially empty. The occurrence of a transition is called an *event* and given an event e, δ_e denotes the corresponding transition. The semantics of a CFSM is defined as follows:

Definition 2 (Semantics): The semantics of a CFSM $\mathcal{T} = \langle L, \ell_0, Q, M, \Sigma, \Delta\rangle$ *is given by an infinite Labelled Transition System (LTS)* $[\![\mathcal{T}]\!] = \langle X, \vec{x}_0, \Sigma, \rightarrow\rangle$*, where* (i) $X \stackrel{\mathbf{def}}{=} L \times (M^*)^{|Q|}$ *is the set of states,* (ii) $\vec{x}_0 \stackrel{\mathbf{def}}{=} \langle \ell_0, \epsilon, \ldots, \epsilon\rangle$ *is the initial state (where* ϵ *is the empty word),* (iii) Σ *is the set of actions, and* (iv) $\rightarrow \stackrel{\mathbf{def}}{=} \bigcup_{\delta\in\Delta} \stackrel{\delta}{\rightarrow} \subseteq X \times \Sigma \times X$ *is the transition relation where* $\stackrel{\delta}{\rightarrow}$ *is defined by:*

$$\frac{\delta = \langle \ell, i!m, \ell'\rangle \in \Delta \qquad w_i' = w_i \cdot m}{\langle \ell, w_1, \ldots, w_i, \ldots, w_{|Q|}\rangle \stackrel{\delta}{\rightarrow} \langle \ell', w_1, \ldots, w_i', \ldots, w_{|Q|}\rangle}$$

$$\frac{\delta = \langle \ell, i?m, \ell'\rangle \in \Delta \qquad w_i = m \cdot w_i'}{\langle \ell, w_1, \ldots, w_i, \ldots, w_{|Q|}\rangle \stackrel{\delta}{\rightarrow} \langle \ell', w_1, \ldots, w_i', \ldots, w_{|Q|}\rangle}$$

where $a.b$ denotes the concatenation of words a and b.

A global state of a CFSM $\mathcal{T}$ is thus a tuple $\langle \ell, w_1, ..., w_{|Q|}\rangle \in X = L \times (M^*)^{|Q|}$ where ℓ is the current location of $\mathcal{T}$ and $w_1, ..., w_{|Q|}$ are finite words on M^* which give the content of the queues in Q. At the beginning, all queues are empty, so the initial state is $\vec{x}_0 = \langle \ell_0, \varepsilon, \cdots, \varepsilon\rangle$. Given a CFSM $\mathcal{T}$, two states $\vec{x}, \vec{x}' \in X$ and an event e,

to simplify the notations we sometimes denote $\vec{x} \xrightarrow{\delta_c} \vec{x}'$ by $\vec{x} \xrightarrow{e} \vec{x}'$. An *execution* of $\mathcal{T}$ is a sequence $\vec{x}_0 \xrightarrow{e_1} \vec{x}_1 \xrightarrow{e_2} \ldots \xrightarrow{e_m} \vec{x}_m$ where $\vec{x}_i \xrightarrow{e_{i+1}} \vec{x}_{i+1} \in \rightarrow \forall i \in [0, m-1]$. Given a set of states $Y \subseteq X$, $\mathsf{Reach}^{\mathcal{T}}_{\Delta'}(Y)$ corresponds to the set of states that are reachable in $[\![\mathcal{T}]\!]$ from Y only triggering transitions of $\Delta' \subseteq \Delta$ in $\mathcal{T}$, whereas $\mathsf{CoReach}^{\mathcal{T}}_{\Delta'}(Y)$ denotes the set of states from which Y is reachable only triggering transitions of Δ'. They can be defined by the following fixpoint equations:

$$\mathsf{Reach}^{\mathcal{T}}_{\Delta'}(Y) \stackrel{\mathbf{def}}{=} \mu B.Y \cup \mathsf{Post}^{\mathcal{T}}_{\Delta'}(B) \quad (1)$$

$$\mathsf{CoReach}^{\mathcal{T}}_{\Delta'}(Y) \stackrel{\mathbf{def}}{=} \mu B.Y \cup \mathsf{Pre}^{\mathcal{T}}_{\Delta'}(B) \quad (2)$$

where $\mu B.f(B)$ denotes the least fixpoint of the function $B \mapsto f(B)$ and:

$$\mathsf{Post}^{\mathcal{T}}_{\Delta'}(Y) \stackrel{\mathbf{def}}{=} \{\vec{x}' \in X | \exists \vec{x} \in Y, \exists \delta \in \Delta' : \vec{x} \xrightarrow{\delta} \vec{x}'\} \quad (3)$$

$$\mathsf{Pre}^{\mathcal{T}}_{\Delta'}(Y) \stackrel{\mathbf{def}}{=} \{\vec{x}' \in X | \exists \vec{x} \in Y, \exists \delta \in \Delta' : \vec{x}' \xrightarrow{\delta} \vec{x}\} \quad (4)$$

Since the function on the right hand side of equations (1) and (2) are continuous, the Knaster-Tarski [23] theorem ensures that the least fixpoint of this function actually exists. Although there is no general algorithm that can exactly compute the (co-)reachability set in our setting [5], there exists some techniques that allow us to compute an overapproximation of this set (see section IV).

Asynchronous Product. A distributed system $\mathcal{T}$ is generally composed of several subsystems $\mathcal{T}_i$ ($\forall i \in [1, n]$) acting in parallel. In fact, $\mathcal{T}$ is defined by a CFSM resulting from the asynchronous (interleaved) product of the n subsystems $\mathcal{T}_i$, also modeled by CFSMs. This can be defined through the asynchronous product of two subsystems.

Definition 3: The asynchronous product of 2 CFSMs $\mathcal{T}_i = \langle L_i, \ell_{0,i}, Q_i, M_i, \Sigma_i, \Delta_i \rangle$ ($i = 1, 2$), denoted by $\mathcal{T}_1 || \mathcal{T}_2$, is defined by a CFSM $\mathcal{T} = \langle L, \ell_0, Q, M, \Sigma, \Delta \rangle$, where (i) $L \stackrel{\mathbf{def}}{=} L_1 \times L_2$, (ii) $\ell_0 \stackrel{\mathbf{def}}{=} \ell_{0,1} \times \ell_{0,2}$, (iii) $Q \stackrel{\mathbf{def}}{=} Q_1 \cup Q_2$, (iv) $M \stackrel{\mathbf{def}}{=} M_1 \cup M_2$, (v) $\Sigma \stackrel{\mathbf{def}}{=} \Sigma_1 \cup \Sigma_2$, and (vi) $\Delta \stackrel{\mathbf{def}}{=} \{\langle \langle \ell_1, \ell_2 \rangle, \sigma_1, \langle \ell'_1, \ell_2 \rangle \rangle | (\langle \ell_1, \sigma_1, \ell'_1 \rangle \in \Delta_1) \wedge (\ell_2 \in L_2)\} \cup \{\langle \langle \ell_1, \ell_2 \rangle, \sigma_2, \langle \ell_1, \ell'_2 \rangle \rangle | (\langle \ell_2, \sigma_2, \ell'_2 \rangle \in \Delta_2) \wedge (\ell_1 \in L_1)\}$.

Note that in the previous definition, Q_1 and Q_2 are not necessarily disjoint; this allows the subsystems to communicate between them via common queues.

Definition 4: [Distributed system] A distributed system $\mathcal{T} = \langle L, \ell_0, Q, M, \Sigma, \Delta \rangle$ is defined by the asynchronous product of n CFSMs $\mathcal{T}_i = \langle L_i, \ell_{0,i}, Q_i, M, \Sigma_i, \Delta_i \rangle$ ($\forall i \in [1, n]$) acting in parallel and exchanging information through FIFO channels.

Note that a distributed system is also modeled by a CFSM, since the asynchronous product of several CFSMs is a CFSM. To avoid the confusion between the model of one process and the model of the whole system, in the sequel, a CFSM $\mathcal{T}_i$ always denotes the model of a single process, and a distributed system $\mathcal{T} = \langle L, \ell_0, Q, M, \Sigma, \Delta \rangle$ always denotes the model of the global system. Below, $\mathcal{T} = \mathcal{T}_1 || \ldots || \mathcal{T}_n$ is the considered distributed system.

Communication Architecture of the System. We consider an architecture for the system $\mathcal{T} = \mathcal{T}_1 || \ldots || \mathcal{T}_n$ defined in Definition 4 with *point-to-point* communication i.e., any subsystem $\mathcal{T}_i$ can send messages to any other subsystem $\mathcal{T}_j$ through a queue $Q_{i,j}$. Thus, only $\mathcal{T}_i$ can write a message m on $Q_{i,j}$ (this is denoted by $Q_{i,j}!m$) and only $\mathcal{T}_j$ can read a message m on this queue (this is denoted by $Q_{i,j}?m$). Moreover, we suppose that the queues are unbounded, that the message transfers between the subsystems are reliable and may suffer from arbitrary non-zero delays, and that no *global clock* or *perfectly synchronized local clocks* are available. With this architecture, the set Q_i of $\mathcal{T}_i$ ($\forall i \in [1, n]$) can be rewritten by $Q_i = \{Q_{i,j}, Q_{j,i} \,|\, (1 \leq j \leq n) \wedge (j \neq i)\}$ and $\forall j \neq i \in [1, n]$, $\Sigma_i \cap \Sigma_j = \emptyset$. Let $\delta_i = \langle \ell_i, \sigma_i, \ell'_i \rangle \in \Delta_i$ be a transition of $\mathcal{T}_i$, $\mathsf{global}(\delta_i) \stackrel{\mathbf{def}}{=} \{\langle \langle \ell_1, \ldots, \ell_{i-1}, \ell_i, \ell_{i+1}, \ldots, \ell_n \rangle, \sigma_i, \langle \ell_1, \ldots, \ell_{i-1}, \ell'_i, \ell_{i+1}, \ldots, \ell_n \rangle \rangle \in \Delta \,|\, \forall j \neq i \in [1, n] : \ell_j \in L_j\}$ is the set of transitions of Δ that can be built from δ_i in $\mathcal{T}$. We extend this definition to sets of transitions $D \subseteq \Delta_i$ of the subsystem $\mathcal{T}_i$: $\mathsf{global}(D) \stackrel{\mathbf{def}}{=} \bigcup_{\delta_i \in D} \mathsf{global}(\delta_i)$. By abuse of notation, we write $\Delta \setminus \Delta_i$ instead of $\Delta \setminus \mathsf{global}(\Delta_i)$ to denote the set of transitions of Δ that are not built from Δ_i.

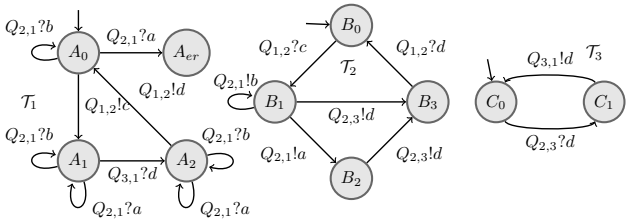

Fig. 2. Running Example

Example 1: Let us illustrate the concepts of distributed system and CFSM with our running example depicted in Figure 2. It models a factory composed of three components $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$. The subsystem $\mathcal{T}_2$ produces two kinds of items, a and b, and sends these items to $\mathcal{T}_1$ to finish the job. At reception, $\mathcal{T}_1$ must immediately terminate the process of each received item. $\mathcal{T}_1$ can receive and process b items at any time, but must be in a turbo mode *to receive and process a items. The subsystem $\mathcal{T}_1$ can therefore be in* normal *mode modeled by the location A_0 or in* turbo *mode (locations A_1 and A_2). In normal mode, if $\mathcal{T}_1$ receives an item a, an error occurs (transition in location A_{er}). Since $\mathcal{T}_1$ cannot always be in turbo mode, a protocol between $\mathcal{T}_1$ and $\mathcal{T}_2$ is imagined. At the beginning, $\mathcal{T}_1$ informs (action $Q_{1,2}!c$) $\mathcal{T}_2$ that it goes in a turbo mode, then $\mathcal{T}_2$ sends a and b items. At the end of a working session, $\mathcal{T}_2$ informs $\mathcal{T}_1$ (action $Q_{2,3}!d$) that it has completed its session, so that $\mathcal{T}_1$ can go back in normal mode. However, this information has to transit through $\mathcal{T}_3$ via queues $Q_{2,3}$ and $Q_{3,1}$, as $\mathcal{T}_3$ must also record this end of session. Since the message d can be transmitted faster than some items a and b, one can easily find a scenario where $\mathcal{T}_1$ decides to go back to A_0 and ends up in the A_{er} location by reading the message a. It is due to the fact that the subsystems cannot observe the content of the queues and thus $\mathcal{T}_1$ does not know if there is a message a in queue $Q_{2,1}$ when it arrives in A_0.*

III. State Avoidance Control Problem

In the sequel, we are interested in the state avoidance control problem which consists in preventing the system from reaching some undesirable states.

A. Control Architecture

The system to be controlled is composed of n subsystems $\mathcal{T}_i$ ($\forall i \in [1,n]$) (see Definition 4). Classically, we want to associate a local controller $\mathcal{C}_i$ ($\forall i \in [1,n]$) with each subsystem $\mathcal{T}_i$ in order to satisfy some control requirements. We suppose that the controllers can communicate with each other by adding some information (e.g., their current state, their state estimates, ...) to the messages exchanged by the subsystems (see Figure 1, where $\mathcal{C}_1$ and $\mathcal{C}_2$ use the queues $Q_{1,2}$ and $Q_{2,1}$ of $\mathcal{T}_1$ and $\mathcal{T}_2$ to exchange information (we duplicate the queues in Figure 1 to emphasize the fact that the controllers can exchange information). These communications allow them to exchange information in order to refine their knowledge about the global state of the system and thus their control policy. Due to the communication delay, the controllers cannot communicate between them synchronously, but we assume that a local controller communicate synchronously with its subsystem. This assumption is realistic, since both generally reside in the same site.

Each controller $\mathcal{C}_i$ interacts with $\mathcal{T}_i$ in a feedback manner: the controller $\mathcal{C}_i$ observes the last action fired by $\mathcal{T}_i$ and computes, from this observation and some information received from the other controllers (corresponding to some state estimates), a set of actions that the subsystem $\mathcal{T}_i$ cannot fire in order to ensure the desired properties on the global system. Following the Ramadge & Wonham's theory [20], the set of actions Σ_i of $\mathcal{T}_i$ is partitioned into the set of controllable actions $\Sigma_{i,c}$, that can be forbidden by $\mathcal{C}_i$, and the set of uncontrollable actions $\Sigma_{i,uc}$, that cannot be forbidden by $\mathcal{C}_i$. The subsets $\Sigma_{1,c}, \ldots, \Sigma_{n,c}$ are disjoint, because $\Sigma_i \cap \Sigma_j = \emptyset$ ($\forall i \neq j \in [1,n]$). In this paper, we assume that inputs are uncontrollable, and outputs are controllable. This is a classical assumption for reactive systems. Our algorithm however does not depend on this particular partition of the actions, since one of its parameters is the set of uncontrollable actions. The set of actions, that can be controlled by at least one controller, is denoted by Σ_c and is defined by $\Sigma_c \stackrel{\mathbf{def}}{=} \bigcup_{i=1}^{n} \Sigma_{i,c}$; the set of actions, that cannot be controlled, is denoted by Σ_{uc} and is defined by $\Sigma_{uc} \stackrel{\mathbf{def}}{=} \Sigma \setminus \Sigma_c = \bigcup_{i=1}^{n} \Sigma_{i,uc}$. This cut also induces a partition on the set of transitions Δ_i into the sets $\Delta_{i,c}$ and $\Delta_{i,uc}$. The set of transitions Δ is similarly partitioned into the sets Δ_c and Δ_{uc}. Note that with our architecture, each local controller $\mathcal{C}_i$ has only a partial observation of the global system, since it can observe neither the subsystems $\mathcal{T}_j$ ($\forall j \neq i \in [1,n]$) nor the content of the FIFO channels.

B. Distributed Controller and Controlled Execution

We would like to base the control decision on the state in which the global system $\mathcal{T}$ is (state-feedback control). Unfortunately, a local controller does not generally know the current state of the global system $\mathcal{T}$, due to its partial observation of the system. We therefore assume that a controller must define its control policy from a state estimate corresponding to its evaluation of the states the system $\mathcal{T}$ can possibly be; this depends on the different observations performed by the controller. It is formally defined as follows:

Definition 5: A local controller $\mathcal{C}_i$ *is a function* $\mathcal{C}_i : 2^X \rightarrow 2^{\Sigma_{i,c}}$ *which defines, for each estimate* $E \in 2^X$ *of the current state of* $\mathcal{T}$ *according to* $\mathcal{C}_i$*, the set of controllable actions that* $\mathcal{T}_i$ *may not execute.*

This definition of a controller does not explain how each local controller can compute a state estimate. In section IV-A, we outline an algorithm that allows $\mathcal{C}_i$ to compute this state estimate during the execution of this system. Note that besides the preciseness of the state estimate, one important property that should be satisfied by the state estimate E computed by a local controller is that the actual current state of the system is contained in E.

Based on Definition 5, a *distributed controller* is defined by:

Definition 6: A distributed controller $\mathcal{C}_{\mathsf{di}}$ *is defined by a tuple* $\mathcal{C}_{\mathsf{di}} \stackrel{\mathbf{def}}{=} \langle \mathcal{C}_i \rangle_{i=1}^{n}$ *where* $\mathcal{C}_i$ ($\forall i \in [1,n]$) *is a local controller.*

The *controlled execution* characterizes the executions that can occur in $\mathcal{T}$ under the control of $\mathcal{C}_{\mathsf{di}}$.

Definition 7: Given a distributed controller $\mathcal{C}_{\mathsf{di}} = \langle \mathcal{C}_i \rangle_{i=1}^{n}$, $s = \vec{x}_0 \xrightarrow{e_1} \vec{x}_1 \xrightarrow{e_2} \ldots \xrightarrow{e_m} \vec{x}_m$ *is a* controlled execution *of* $\mathcal{T}$ *under the control of* $\mathcal{C}_{\mathsf{di}}$ *if* $\forall k \in [1,m]$*, whenever* $\delta_{e_k} = \langle \ell_i, \sigma_{e_k}, \ell_i' \rangle \in \Delta_i$ *and the estimate of* $\mathcal{C}_i$ *of the current state* $\vec{x}_{k-1}$ *of* $\mathcal{T}$ *is* E*,* $\sigma_{e_k} \notin \mathcal{C}_i(E)$.

Note that with this definition, the language of the controlled system is controllable with respect to the language of the original system. It is basically due to the fact that each local controller is only able to disable the controllable actions that can occur in its corresponding subsystem.

C. Definition of the Control Problem

The aim of control synthesis is to constraint a system to satisfy a goal property. The goal properties we consider are invariance properties, defined by a subset $I \subseteq X$ of states, in which any execution of the transition system should be confined. Alternatively, it can be viewed as a state avoidance property $Bad = X \setminus I$, which defines a set of states that no execution should reach. Notice that the specification Bad can involve the contents of the FIFO channels (recall that $X = L \times (M^*)^{|Q|}$). We define the problem as follows:

Problem 1 (Distributed State Avoidance Control Problem): Given a set of forbidden states $Bad \subseteq X$*, the* distributed state avoidance control problem *(the* distributed problem *for short) consists in synthesizing a distributed controller* $\mathcal{C}_{\mathsf{di}} = \langle \mathcal{C}_i \rangle_{i=1}^{n}$ *such that each controlled execution of the system* $\mathcal{T}$ *under the control of* $\mathcal{C}_{\mathsf{di}}$ *avoids* Bad*.*

Proposition 1: The existence of a solution for the distributed problem is an undecidable problem.

Intuitively, this result holds because even if we consider the most trivial solution (i.e., each local controller forbids all its controllable actions), we cannot determine if it prevents from reaching Bad, as (co-)reachability is undecidable in the model of CFSM [5]. According to Proposition 1, an

algorithm cannot always compute a distributed controller $\mathcal{C}_{\mathsf{di}}$ for the distributed problem (when a solution exists) and ensure the termination of the computations. Hence, our approach uses approximations to ensure the termination of the computations and our aim is to find, in this way, solutions that are correct and that restrict as less as possible the behavior of the system $\mathcal{T}$.

IV. Computation of Distributed Controllers for the Distributed Problem

In this section, we define a semi-algorithm for the distributed control problem. But first, we outline the algorithm used by the local controller allowing to compute the state estimates.

A. State Estimates Algorithm

As described in Definition 7, the local controllers $\mathcal{C}_i$ define their control policy from an estimate of the current state of the system $\mathcal{T}$ that they compute during its execution. In this section, we informally explain the SE-algorithm that allows each local controller to compute its own state estimate. More details can be found in [12].

Vector Clocks. This algorithm is based on the use of vector clocks that allow the controllers to have a better understanding of the concurrent execution of the distributed system $\mathcal{T}$ by determining the causal and temporal relationship between the events occurring in the execution of $\mathcal{T}$. To compute these vector clocks, we use Mattern's algorithm [18]. In this algorithm, each subsystem $\mathcal{T}_i$ ($\forall i \in [1,n]$) has a vector clock $V_i \in \mathbb{N}^n$ of width n and each element $V_i[j]$ ($\forall j \in [1,n]$) is a counter which represents the knowledge of $\mathcal{T}_i$ regarding the past events of T_j: if $V_i[j] = 3$, it means that $\mathcal{T}_i$ knows that at least 3 events occurred in $\mathcal{T}_j$. Each time an event occurs in a subsystem $\mathcal{T}_i$, the vector clock V_i is updated to take into account the occurrence of this event (see [18] for more details). When $\mathcal{T}_i$ sends a message to some subsystem $\mathcal{T}_j$, this vector clock is piggybacked and allows $\mathcal{T}_j$, after reception, to update its own vector clock.

Online State Estimates. The local controllers compute their state estimates online: each time an event occurs in $\mathcal{T}_i$, its controller $\mathcal{C}_i$ updates its state estimate E_i and its vector clock V_i from the information received from $\mathcal{T}_i$. In order to have better state estimates, we also assume that the controllers can communicate with each other by adding some information (vector clocks and state estimates) to the messages exchanged by the subsystems.

State Estimates Based on Reachability Sets. Now, we explain how the SE-algorithm allows the local controllers to maintain online their state estimates. At the beginning of the execution of $\mathcal{T}$, each FIFO channel is empty and each $\mathcal{T}_i$ is in its initial location $\ell_{i,0}$. So, $\vec{x}_0$ is known by each $\mathcal{C}_i$. However, a subsystem $\mathcal{T}_j$ may start its execution, while $\mathcal{T}_i$ is still in its initial location, and therefore $\mathcal{C}_i$ must take into account all the global states that are reachable by taking the transitions of the other subsystems $\mathcal{T}_j$. So, the initial estimate E_i of $\mathcal{C}_i$ is this set of reachable global states i.e., $E_i = \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\vec{x}_0)$. Next, each time a subsystem $\mathcal{T}_i$ executes a transition δ, its controller $\mathcal{C}_i$ updates E_i as follows:

- **Output:** if δ sends message m to $\mathcal{T}_j$, the new estimate of $\mathcal{C}_i$ is given by the set of states that are reachable from E_i by firing the transition δ followed by transitions in $\Delta \setminus \Delta_i$ i.e., $E_i = \mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(\mathsf{Post}^{\mathcal{T}}_{\delta}(E_i))$. Indeed, after the occurrence of δ, the other subsystems $\mathcal{T}_k$ ($\forall k \neq i \in [1,n]$) could read and write on their queues. Next, $\mathcal{C}_i$ updates its vector clock according to Mattern's algorithm. Finally, m is sent to $\mathcal{T}_j$ with E_i and V_i.
- **Input:** if δ reads message m sent by $\mathcal{T}_j$ and tagged with E_j and V_j, then the new state estimate of $\mathcal{C}_i$ is computed from its state estimate E_i and from E_j in the following way. First, E_i is updated to take into account the occurrence of δ; this update corresponds to the set of states reachable from E_i by δ i.e., $E_i = \mathsf{Post}^{\mathcal{T}}_{\delta}(E_i)$. Next, the estimate E_j (sent by $\mathcal{T}_j$) is updated by $\mathcal{C}_i$ to contain the current state of $\mathcal{T}$. This update actually depends on the values of the vector clocks V_i and V_j that allow $\mathcal{C}_i$ to guess the possible behaviors of $\mathcal{T}$ between the sending and the reception of m. For example, if $V_i[i] = V_j[i]$, then $\mathcal{C}_i$ knows that $\mathcal{T}_i$ has executed no action between the sending and the reception of m and hence it updates E_j in the following way: $E_j = \mathsf{Post}^{\mathcal{T}}_{\delta}(\mathsf{Reach}^{\mathcal{T}}_{\Delta\setminus\Delta_i}(E_i))$. In [12], we give the various updates of E_j w.r.t. the values of V_i and V_j. Then, the new estimate of $\mathcal{C}_i$ is given by the intersection of the updates of E_i and E_j. Finally, the vector clock V_i is updated according to Mattern's algorithm.

One can note that the SE-algorithm is based on the computation of reachability sets that cannot always be computed for undecidability reasons. In section V, we explain how to overcome this obstacle by using abstract interpretation techniques to compute an overapproximation of the reachability sets in a finite number of steps. Moreover, the SE-algorithm satisfies the important property of completeness [12], which means that the current global state of $\mathcal{T}$ is always included in the state estimates computed by the local controllers $\mathcal{C}_i$.

B. Semi-Algorithm for the Distributed Problem

Our algorithm, which synthesizes a distributed controller $\mathcal{C}_{\mathsf{di}}$ for the distributed problem, is composed of two parts:

- **Offline part:** we compute the set $I(Bad)$ of states of the global system $\mathcal{T}$ that can lead to Bad by a sequence of uncontrollable transitions. Next, we compute, for each local controller $\mathcal{C}_i$, a control function $\mathcal{F}_i$ which gives, for each action σ of $\mathcal{T}_i$, the set of states of $\mathcal{T}$ that can lead to $I(Bad)$ by a transition labeled by σ. This information is used by $\mathcal{C}_i$, in the online part, to define its control policy.
- **Online part:** During the execution of the system $\mathcal{T}$, each local controller $\mathcal{C}_i$ uses the SE-algorithm to obtain its own state estimate E_i and computes from this information the actions to be forbidden.

These two parts are formalized as follows.

Offline Part. The set $I(Bad)$ of states of $\mathcal{T}$ leading uncontrollably to Bad is given by $\mathsf{CoReach}^{\mathcal{T}}_{\Delta_{uc}}(Bad)$ which cannot always be computed, because coreachability is undecidable for CFSM. So, we make the use of abstract interpretation techniques to overapproximate it (see section V).

Next, we define, for each local controller $\mathcal{C}_i$, the control function $\mathcal{F}_i : \Sigma_i \times 2^X \to 2^X$, which gives, for each action $\sigma \in \Sigma_i$ and set $B \subseteq X$ of states to be forbidden, the set $\mathcal{F}_i(\sigma, B)$ of global states in which the action σ must be forbidden. This set corresponds, more precisely, to the greatest set $\mathcal{O}$ of states of $\mathcal{T}$ such that, for each state $\vec{x} \in \mathcal{O}$, there exists a transition labeled by σ leading to B from $\vec{x}$:

$$\mathcal{F}_i(\sigma, B) \stackrel{\text{def}}{=} \begin{cases} \mathsf{Pre}^{\mathcal{T}}_{\mathsf{Trans}(\sigma)}(B) & \text{if } \sigma \in \Sigma_{i,c} \\ \emptyset & \text{otherwise} \end{cases} \quad (5)$$

where $\mathsf{Trans}(\sigma)$ denotes the set of transitions labeled by σ. We compute, for each transition $\sigma \in \Sigma_i$, the set $\mathcal{F}_i(\sigma, I(Bad))$ ($\forall i \in [1, n]$). This information is used, during the execution of the system $\mathcal{T}$, by the local controller $\mathcal{C}_i$ to compute the actions to be forbidden.

Online Part. The local controller $\mathcal{C}_i$ is formally defined, for each state estimate $E \in 2^X$, by:

$$\mathcal{C}_i(E) \stackrel{\text{def}}{=} \{\sigma \in \Sigma_i \mid \mathcal{F}_i(\sigma, I(Bad)) \cap E \neq \emptyset\} \quad (6)$$

Thus, if E is the state estimate of $\mathcal{C}_i$, it forbids an action $\sigma \in \Sigma_i$ if and only if there exists a state $\vec{x} \in E$ in which the action σ must be forbidden in order to prevent the system $\mathcal{T}$ from reaching $I(Bad)$ (i.e., $\exists \vec{x} \in E : \vec{x} \in \mathcal{F}_i(\sigma, I(Bad))$).

During the execution of the system, when the subsystem $\mathcal{T}_i$ ($\forall i \in [1, n]$) executes a transition $\delta = \langle \ell_i, \sigma, \ell'_i \rangle$, the local controller $\mathcal{C}_i$ receives the following information:

- if $\sigma = Q_{j,i}?m$ (with $j \neq i \in [1, n]$), it receives the action σ, and the pair $\langle E_j, V_j \rangle$ tagging m.
- if $\sigma = Q_{i,j}!m$ (with $j \neq i \in [1, n]$), it receives σ.

In both cases, since $\mathcal{C}_i$ knows that $\mathcal{T}_i$ was in the location ℓ_i before triggering σ, this controller can infer the fired transition. $\mathcal{C}_i$ then uses the SE-algorithm with this information to update its state estimate E_i and computes, from this state estimate, the set $\mathcal{C}_i(E_i)$ of actions that $\mathcal{T}_i$ cannot execute.

The following theorem proves that this algorithm synthesizes correct controllers for the distributed problem.

Theorem 1: Given a set of forbidden states $Bad \subseteq X$, our distributed controller $\mathcal{C}_{\mathsf{di}} = \langle \mathcal{C}_i \rangle_{i=1}^n$ solves the distributed problem if $\vec{x}_0 \notin I(Bad)$.

Proof: We prove by induction on the length m of the sequences of transitions (these sequences begin in the initial state) that $I(Bad)$ is not reachable in the system $\mathcal{T}$ under the control of $\mathcal{C}_{\mathsf{di}}$, which implies that Bad is not reachable, because $Bad \subseteq I(Bad)$:

Base case ($m = 0$): Since $\vec{x}_0 \notin I(Bad)$, the execution of the system $\mathcal{T}$ under the control of $\mathcal{C}_{\mathsf{di}}$ starts in a state which does not belong to $I(Bad)$.

Induction step: we suppose that the proposition holds for the sequences of transitions of length less than or equal to m and we prove that this property remains true for the sequences of transitions of length $m+1$. By induction hypothesis, each state $\vec{x}_1$ reachable by a sequence of transitions of length m does not belong to $I(Bad)$ and we show that each transition $\delta \in \Delta$, which can lead to a state $\vec{x}_2 \in I(Bad)$ from this state $\vec{x}_1$ in $\mathcal{T}$, cannot be fired from $\vec{x}_1$ in the system $\mathcal{T}$ under the control of $\mathcal{C}_{\mathsf{di}}$. For that, we consider two cases and we suppose that δ is executed by $\mathcal{T}_i$ and is labeled by σ:

- if δ is controllable, then σ is forbidden by $\mathcal{C}_i$ in $\vec{x}_1$ and hence δ cannot be fired from $\vec{x}_1$. Indeed, the estimate E_i of $\mathcal{C}_i$ contains $\vec{x}_1$, because the SE-algorithm is complete. Moreover, we have that $\vec{x}_1 \in \mathcal{F}_i(\sigma, I(Bad))$, because $\vec{x}_1 \in \mathsf{Pre}^{\mathcal{T}}_{\delta}(\vec{x}_2)$ and $\vec{x}_2 \in I(Bad)$. Therefore, $\sigma \in \mathcal{C}_i(E_i)$ (by (6)), which implies that δ cannot be fired from $\vec{x}_1$.
- if δ is uncontrollable, since $\vec{x}_2 \in I(Bad)$, so is $\vec{x}_1$ because of the definition of $I(Bad)$, which is impossible by hypothesis.

Hence, in the system $\mathcal{T}$ under the control of $\mathcal{C}_{\mathsf{di}}$, the forbidden state $\vec{x}_2$ cannot be reached from $\vec{x}_1$ by the transition δ. ■

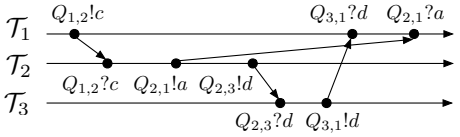

Fig. 3. An execution of the running example.

Example 2: We consider the sequence of actions of our running example (see Example 1) depicted in Figure 3. A state of the global system is denoted by $\langle \ell_1, \ell_2, \ell_3, w_{1,2}, w_{2,1}, w_{2,3}, w_{3,1} \rangle$ where ℓ_i is the location of $\mathcal{T}_i$ (for $i = 1, 2, 3$) and $w_{1,2}$, $w_{2,1}$, $w_{2,3}$ and $w_{3,1}$ denote the content of the queues $Q_{1,2}$, $Q_{2,1}$, $Q_{2,3}$ and $Q_{3,1}$. The set Bad is given by the set of global states where the location of $\mathcal{T}_1$ is A_{er}. Thus, $I(Bad) = Bad \cup \{\langle \ell_1, \ell_2, \ell_3, w_{1,2}, w_{2,1}, w_{2,3}, w_{3,1} \rangle | (\ell_1 = A_0) \wedge (w_{2,1} = a.M^*)\}$. At the beginning of the execution of $\mathcal{T}$, the state estimates of the subsystems are $E_1 = \{\langle A_0, B_0, D_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle\}$, $E_2 = \{\langle A_0, B_0, D_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_0, D_0, c, \epsilon, \epsilon, \epsilon \rangle\}$, and $E_3 = \{\langle A_0, B_0, D_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_0, D_0, c, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_1, D_0, \epsilon, b^*, \epsilon, \epsilon \rangle, \langle A_1, B_2, D_0, \epsilon, b^*(a + \epsilon), \epsilon, \epsilon \rangle, \langle A_1, B_3, D_0, \epsilon, b^*(a + \epsilon), d, \epsilon \rangle\}$. After the first transition $\langle A_0, Q_{1,2}!c, A_1 \rangle$, the state estimate of the controller $\mathcal{C}_1$ is not really precise, because a lot of things may happen without the controller $\mathcal{C}_1$ being informed: $E_1 = \{\langle A_1, B_0, D_0, c, \epsilon, \epsilon, \epsilon \rangle, \langle A_1, B_1, D_0, \epsilon, b^*, \epsilon, \epsilon \rangle, \langle A_1, B_2, D_0, \epsilon, b^*a, \epsilon, \epsilon \rangle, \langle A_1, B_3, D_0, \epsilon, b^*(a+\epsilon), d, \epsilon \rangle, \langle A_1, B_3, D_1, \epsilon, b^*(a+\epsilon), \epsilon, \epsilon \rangle, \langle A_1, B_3, D_0, \epsilon, b^*(a + \epsilon), \epsilon, d \rangle\}$. However, after the second transition $\langle B_0, Q_{1,2}?c, B_1 \rangle$, the controller $\mathcal{C}_2$ has an accurate state estimate: $E_2 = \{\langle A_1, B_1, D_0, \epsilon, \epsilon, \epsilon, \epsilon \rangle\}$. We skip a few steps and consider the state estimates before the sixth transition $\langle D_1, Q_{3,1}!d, D_0 \rangle$: E_1 is still the same, because the subsystem $\mathcal{T}_1$ did not perform any action, $E_3 = \{\langle A_1, B_3, D_1, \epsilon, b^*(a + \epsilon), \epsilon, \epsilon \rangle\}$, and we do not give E_2, because $\mathcal{T}_2$ is no longer involved. When $\mathcal{T}_3$ sends message d to $\mathcal{T}_1$, it tags it with E_3. Thus, $\mathcal{C}_1$ knows, after receiving this message, that there may be a message a in the queue $Q_{2,1}$. It thus disables the action $A_2 \stackrel{Q_{1,2}!d}{\to} A_0$, as long as this message a is not read (action $A_2 \stackrel{Q_{2,1}?a}{\to} A_2$), to prevent the system from reaching the forbidden states. Note that if we consider the sequence of actions of Figure 3 without the sending and the reception of the message a, then when $\mathcal{T}_1$ reaches the location A_2 by executing the action $Q_{3,1}?d$, its controller $\mathcal{C}_1$ enables the actions $Q_{1,2}!d$, because it knows that there is no message a in $Q_{2,1}$. ◇

V. Effective Algorithm for the Distributed Problem

The algorithms described in the previous sections require the computation of (co-)reachability operators. Those operators cannot be computed exactly because of undecidability reasons. Abstract interpretation-based techniques [6] allows us to compute, in a finite number of steps, an *overapproximation* of the (co-)reachability operators and thus of the set $I(Bad)$ and of the state estimates E_i.

Computation of Reachability Sets by the Means of Abstract Interpretation. For a given set of global states $X' \subseteq X$ and a given set of transitions $\Delta' \subseteq \Delta$, the reachability set from X' can be characterized by the least fixpoint $\mathsf{Reach}^{\mathcal{T}}_{\Delta'}(X') = \mu Y.F_{\Delta'}(Y)$ with $F_{\Delta'}(Y) = X' \cup \mathsf{Post}^{\mathcal{T}}_{\Delta'}(Y)$ (see (1)). Abstract interpretation provides a theoretical framework to compute efficient overapproximation of such fixpoints. The concrete domain i.e., the sets of states 2^X, is substituted by a simpler abstract domain Λ, linked by a *Galois connection* $2^X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \Lambda$ [6], where α (resp. γ) is the abstraction (resp. concretization) function. The fixpoint equation is transposed into the abstract domain. So, the equation to solve has the form: $\lambda = F^{\sharp}_{\Delta'}(\lambda)$, with $\lambda \in \Lambda$ and $F^{\sharp}_{\Delta'} \sqsupseteq \alpha \circ F_{\Delta'} \circ \gamma$ where $\sqsupseteq$ is the comparison operator in the abstract lattice. In that setting, a standard way to ensures that this fixpoint computation converges after a finite number of steps to some overapproximation λ_∞, is to use a *widening operator* ∇. The concretization $c_\infty = \gamma(\lambda_\infty)$ is an overapproximation of the least fixpoint of the function $F_{\Delta'}$.

Choice of the Abstract Domain. In abstract interpretation-based techniques, the quality of the approximation we obtain depends on the choice of the abstract domain Λ. In our case, the main issue is to abstract the content of the FIFO channels. Since the CFSM model is Turing-powerful, the language which represents all the possible contents of the FIFO channels may be recursively enumerable. As discussed in [14], a good candidate, to abstract the contents of the queues is to use the class of regular languages, which can be represented by finite automata. Let us recall the main ideas of this abstraction.

Finite Automata as an Abstract Domain. We first assume that there is only one queue in the distributed system $\mathcal{T}$; we explain later how to handle a distributed system with several queues. With one queue, the concrete domain of the system $\mathcal{T}$ is defined by $X = 2^{L \times M^*}$. A set of states $Y \in 2^{L \times M^*}$ can be viewed as a map $Y : L \mapsto 2^{M^*}$ that associates a language $Y(\ell)$ with each location $\ell \in L$; $Y(\ell)$ therefore represents the possible contents of the queue in the location ℓ. In order to simplify the computation, we substitute the concrete domain $\langle L \mapsto 2^{M^*}, \subseteq \rangle$ by the abstract domain $\langle L \mapsto \mathrm{Reg}(M), \sqsubseteq \rangle$, where $\mathrm{Reg}(M)$ is the set of *regular languages* over the alphabet M and $\sqsubseteq$ denotes the natural extension of the set inclusion to maps. This substitution consists thus in abstracting, for each location, the possible contents of the queue by a regular language. Regular languages have a canonical representation given by finite automata, and each operation (union, intersection, left concatenation,...) in the abstract domain can be performed on finite automata.

Widening Operator. With our abstraction, the widening operator we use to ensure the convergence of the computation, is also performed on a finite automaton, and consists in quotienting the nodes[1] of the automaton by the *k-bounded bisimulation relation* $\equiv_k$; $k \in \mathbb{N}$ is a parameter which allows us to tune the precision, since increasing k improves the quality of the abstractions in general. Two nodes are equivalent w.r.t. $\equiv_k$ if they have the same outgoing path (sequence of labeled transitions) up to length k. While we merge the equivalent nodes, we keep all transitions and we obtain an automaton recognizing a larger language. Note that for a fixed k, the class of automata which results from such a quotient operation from any original automaton, is finite and its cardinality is bounded by a number which is only function of k. This is the reason why when we apply this widening operator regularly, the fixpoint computation terminates (see [14] for more details).

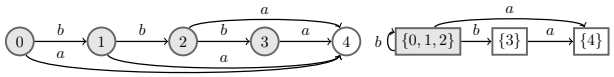

Fig. 4. Illustration of the 1-bounded bisimulation relation $\equiv_1$ for $\mathcal{A}$.

Example 3: We consider $\mathcal{A}$ depicted in Figure 4, whose recognized language is $a + ba + bba + bbba$. We consider the 1-bounded bisimulation relation i.e., two nodes of the automaton are equivalent if they have the same outgoing transitions. So, nodes $0, 1, 2$ are equivalent, since they all have two transitions labeled by a and b. Nodes 3 and 4 are equivalent to no other node since 4 has no outgoing transition whereas only a is enabled in node 3. When we quotient $\mathcal{A}$ by this equivalent relation, we obtain the automaton on the right of Figure 4, whose recognized language is b^*a. $\diamond$

When the system contains several queues $Q = \{Q_1, \ldots, Q_r\}$, their content can be represented by a concatenated word $w_1 \sharp \ldots \sharp w_r$ with one w_i for each queue Q_i and $\sharp$, a delimiter. With this encoding, we represent a set of queue contents by a finite automaton of a special kind, namely a QDD [4]. Since QDDs are finite automata, classical operations (union, intersection, left concatenation,...) in the abstract domain are performed as previously. We must only use a slightly different widening operator not to merge the different queue contents [14].

Effective Algorithm. The Reach and CoReach operators are computed using those abstract interpretation techniques: we proceed to an iterative computation in the abstract domain of regular languages and the widening operator ensures that this computation terminates after a finite number of steps [6]. So the Reach and CoReach operators always give an overapproximation of the (co-)reachable states, whatever the distributed system is. Finally, we define the distributed controller as in section IV-B by using the overapproximations $I'(Bad)$ and E'_i instead of $I(Bad)$ and E_i.

[1]The states of an automaton representing the queue contents are called nodes to avoid the confusion with the states of a CFSM.

VI. Experiments

The control algorithm of this paper is implemented and is available as a module of the McScM tool [19]. This tool accepts a CFSM model of the system as input. The set Bad is given by a set of locations and regular expressions describing what the queues should not contain. Our tool first computes an overapproximation of $I(Bad)$ according to the algorithms of sections IV and V. Then it starts an interactive simulation of the system. At each step, it displays the current state of the system, the transitions forbidden by the controller and asks the user to choose a transition among the allowed ones. Then, it updates the current state of the system and the state estimates as in section IV-B and thus enables or disables the controllable transitions.

Experiment on the Running Example. On this example, our software computes the exact set $I(Bad)$ (see Example 2) if we set the widening parameter $k = 1$. We considered the sequences of events of Example 2 and the software gave the same results as the theory. The computation of $I(Bad)$ and execution of each sequence of events took less than 0.4s of run time and required 1.22 MB of memory on a standard laptop.

Experiment on the Connection/disconnection Protocol. In this example taken from [14], an error occurs when the client and the server send close/disconnect message at the same time. Our controller solves the problem by not allowing the server to send disconnection messages. This experiment took less than 0.1s of run time and required 1.22 MB of memory.

Simulation. Instead of asking the user what transitions should be taken, our software can randomly choose them. We give here the results on the two previous examples, for a 100-steps random run of the system.

example	# subsystems	# channels	time [s]	memory [MB]
running example	3	4	7.13	5.09
c/d protocol	2	2	0.76	1.22

VII. Conclusion and further works

We propose in this paper a novel framework for the control of distributed systems modeled as communicating finite state machines with reliable unbounded FIFO channels. In this framework, each local controller can only observe its subsystem but can communicate with the other controllers by piggybacking extra information, such as state estimates, to the messages sent in the FIFO channels. Our algorithm synthesizes the local controllers that restrict the behavior of a distributed system in order to satisfy a global state avoidance property, e.g. to ensure that an error state is no longer reachable or to bound the size of the FIFO channels. We abstract the content of the FIFO channels by the same regular representation as in [14]; this abstraction leads to a safe effective algorithm. Our experiments show that our approach is tractable and allows a precise control.

As a further work, more elaborate examples must be taken and other techniques must be proposed to minimize the information transmitted for our estimate evaluation algorithm.

References

[1] G. Barrett and S. Lafortune. Decentralized supervisory control with communicating controllers. *IEEE Trans. on Automatic Control*, 45(9):1620–1638, 2000.

[2] S. Bensalem, M. Bozga, S. Graf, D. Peled, and S. Quinton. Methods for knowledge based controlling of distributed systems. In *ATVA'10*, volume 6252 of *LNCS*, pages 52–66. Springer, 2010.

[3] G. Berry and G. Gonthier. The esterel synchronous programming language: Design, semantics, implementation. *Sci. Comput. Program.*, 19(2):87–152, 1992.

[4] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of qdds. In *SAS '97*, pages 172–186, 1997.

[5] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[6] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL'77*, pages 238–252, 1977.

[7] P. Darondeau. Distributed implementations of Ramadge-Wonham supervisory control with petri nets. In *44th IEEE CDC*, pages 2107–2112, 2005.

[8] P. Gastin, N. Sznajder, and M. Zeitoun. Distributed synthesis for well-connected architectures. *Formal Methods in System Design*, 34(3):215–237, 2009.

[9] B. Gaudin and H. marchand. An efficient modular method for the control of concurrent discrete event systems: A language-based approach. *Discrete Event Dynamic System*, 17(2):179–209, 2007.

[10] B. Genest. On implementation of global concurrent systems with local asynchronous controllers. In *CONCUR*, volume 3653 of *LNCS*, pages 443–457, 2005.

[11] S. Jiang and R. Kumar. Decentralized control of discrete event systems with specializations to local control and concurrent systems. *IEEE Trans. on Systems, Man, and Cybernetics*, 30(5):653–660, 2000.

[12] G. Kalyon, T. Le Gall, H. Marchand, and T. Massart. Global state estimates for distributed systems. In *31th IFIP International Conference on FORmal TEchniques for Networked and Distributed Systems, FORTE*, volume 6722 of *LNCS*, pages 198–212, June 2011.

[13] J. Komenda and J.H. van Schuppen. Supremal sublanguages of general specification languages arising in modular control of discrete-event systems. In *44th IEEE CDC*, pages 2775–2780, 2005.

[14] T. Le Gall, B. Jeannet, and T. Jéron. Verification of communication protocols using abstract interpretation of fifo queues. In *AMAST '06*, volume 4019 of *LNCS*, pages 204–219, 2006.

[15] S.-H. Lee and Wong K.C. Structural decentralized control of concurrent discrete-event systems. *Europ. Journal of Control*, 8(5), 2002.

[16] F. Lin, K. Rudie, and S. Lafortune. Minimal communication for essential transitions in a distributed discrete-event system. *IEEE Trans. on Automatic Control*, 52(8):1495–1502, June 2007.

[17] T. Massart. A calculus to define correct tranformations of lotos specifications. In *FORTE*, volume C-2 of *IFIP Transactions*, pages 281–296, 1991.

[18] F. Mattern. Virtual time and global states of distributed systems. In *Proceedings of the Workshop on Parallel and Distributed Algorithms*, pages 215–226, North-Holland / Elsevier, 1989.

[19] McScM, a Model Checker for Symbolic Communicating Machines - version 0.02control. http://altarica.labri.fr/forge/projects/mcscm/wiki/.

[20] P.J. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems*, 77(1):81–98, 1989.

[21] K. Ricker, L. Rudie. Know means no: Incorporating knowledge into discrete-event control systems. *IEEE Trans. on Automatic Control*, 45(9):1656–1668, September 2000.

[22] K. Rudie and W.M. Wonham. Think globally, act locally: decentralized supervisory control. *IEEE Transaction on Automatic Control*, 31(11):1692–1708, November 1992.

[23] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[24] S. Tripakis. Decentralized control of discrete event systems with bounded or unbounded delay communication. *IEEE Trans. on Automatic Control*, 49(9):1489–1501, 2004.

[25] S. Xu and R. Kumar. Distributed state estimation in discrete event systems. In *ACC'09: Proceedings of the 2009 conference on American Control Conference*, pages 4735–4740. IEEE Press, 2009.

[26] T. Yoo and S. Lafortune. A general architecture for decentralized supervisory control of discrete-event systems. *Discrete Event Dynamic Systems*, 12(3):335–377, 2002.